2122

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

7590 07/09/2004

ATTEN: DAVID E. BOUNDY
WILLKIE, FARR & GALLAGHER, LLP
787 SEVENTH AVE.
NEW YORK, NY 10019

RECEIVED
JUL 2 0 2004
Technology Center 2100

EXAMINER	
DAS, CHAMELI	
ART UNIT	PAPER NUMBER
2122	

DATE MAILED: 07/09/2004

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/322,443	05/28/1999	DAVID L. REESE	005231/06-4003	8015

TITLE OF INVENTION: PROFILING OF COMPUTER PROGRAMS EXECUTING IN VIRTUAL MEMORY SYSTEMS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	10/12/2004

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE REFLECTS A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE APPLIED IN THIS APPLICATION. THE PTOL-85B (OR AN EQUIVALENT) MUST BE RETURNED WITHIN THIS PERIOD EVEN IF NO FEE IS DUE OR THE APPLICATION WILL BE REGARDED AS ABANDONED.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL should be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). Even if the fee(s) have already been paid, Part B - Fee(s) Transmittal should be completed and returned. If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

or Fax **(703) 746-4000**

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

7590 07/09/2004

ATTEN: DAVID E. BOUNDY
WILLKIE, FARR & GALLAGHER, LLP
787 SEVENTH AVE.
NEW YORK, NY 10019

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (703) 746-4000, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/322,443	05/28/1999	DAVID L. REESE	005231/06-4003	8015

TITLE OF INVENTION: PROFILING OF COMPUTER PROGRAMS EXECUTING IN VIRTUAL MEMORY SYSTEMS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	10/12/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
DAS, CHAMELI	2122	717-130000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list

(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,

(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 ____________
2 ____________
3 ____________

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent); ❑ individual ❑ corporation or other private group entity ❑ government

4a. The following fee(s) are enclosed:

❑ Issue Fee
❑ Publication Fee (No small entity discount permitted)
❑ Advance Order - # of Copies ____________

4b. Payment of Fee(s):

❑ A check in the amount of the fee(s) is enclosed.
❑ Payment by credit card. Form PTO-2038 is attached.
❑ The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number ____________ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)

❑ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27. ❑ b. Applicant is not claiming SMALL ENTITY status. See, e.g., 37 CFR 1.27(g)(2).

The Director of the USPTO is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

(Authorized Signature) (Date)

TRANSMIT THIS FORM WITH FEE(S)

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/322,443	05/28/1999	DAVID L. REESE	005231/06-4003	8015

7590 07/09/2004

ATTEN: DAVID E. BOUNDY
WILLKIE, FARR & GALLAGHER, LLP
787 SEVENTH AVE.
NEW YORK, NY 10019

EXAMINER	
DAS, CHAMELI	
ART UNIT	PAPER NUMBER
2122	

DATE MAILED: 07/09/2004

Determination of Patent Term Extension under 35 U.S.C. 154 (b)
(application filed after June 7, 1995 but prior to May 29, 2000)

The Patent Term Extension is 0 day(s). Any patent to issue from the above-identified application will include an indication of the 0 day extension on the front page.

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Extension is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (703) 305-1383. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.

Notice of Allowability	Application No. 09/322,443	Applicant(s) REESE ET AL. [illegible]	
	Examiner CHAMELI C. DAS	Art Unit 2122	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *the amendment filed on 5/28/04*.
2. ☒ The allowed claim(s) is/are *1-81*.
3. ☒ The drawings filed on *28 December 2000* are accepted by the Examiner.
4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

5. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.
6. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
   1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).
7. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☒ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☒ Information Disclosure Statements (PTO-1449 or PTO/SB/08), Paper No./Mail Date 6/28/04
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application (PTO-152)
6. ☐ Interview Summary (PTO-413), Paper No./Mail Date _____ .
7. ☐ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

1. This action is in response to the amendment filed on 5/24/04.

2. Claims 1, 16, and 67 have been amended.

3. Formal drawings and a preliminary amendment submitted on 12/28/2000 have been acknowledged.

4. Claims 1-81 have been allowed.

REASON FOR ALLOWANCE

5. The following is an examiner's statement of reason for allowance:

The cited prior art taken alone or in combination fail to teach, in combination with the other claimed limitations, a method for recording profile information that records physical memory addresses referenced during an execution interval of the program, the profile information being distinct from address translation information for use by the address translation circuit, as recited in the independent claims 1, 16 and 67.

Conclusion

6. The prior art made or record and not relied upon is considered pertinent to applicant's disclosure.

TITLE: Collection of timing and coverage data through a debugging interface, US 6721941 B1

TITLE: Method and apparatus for multiple application trace streams, US 6708173 B1

TITLE: Dynamic optimizing object code translator for architecture emulation and dynamic optimizing object code translation method, US 6463582 B1

TITLE: Facilities for detailed software performance analysis in a multithreaded processor, US 6256775 B1

TITLE: Software HLR architecture, US 6351646 B1

TITLE: Prefetching data using profile of cache misses from earlier code executions, US 6157993

TITLE: Information processing system and information processing method, US 6553431 B1.

TITLE: The Profile Naming Service, author: Peterson, ACM, 1988.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chameli Das whose telephone number is 703-305-1339.

The examiner can normally be reached on Monday-Friday from 7:00 A.M. to 3:30 P.M.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor Tuan Dam can be reached at 703-305-4552. The fax number for this group is (703) 872-9306.

An inquiry of general nature or relating to the status of this application or proceeding should be directed to the group receptionist whose telephone number is 703-305-9600.

Chameli C. Das

CHAMELI C. DAS
PRIMARY EXAMINER

6/28/04

Sheet 1 of 1

FORM PTO-1449 U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE **INFORMATION DISCLOSURE CITATION** (Use several sheets if necessary)	ATTY. DOCKET NO. 005231/06-4003
	SERIAL NO. 09/322,443
	APPLICANT David L. Reese, et al.
	FILING DATE May 28, 1999
	GROUP ART UNIT 2122

OIPE JC33 MAY 0 6 2003 PATENT & TRADEMARK OFFICE

U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING DATE IF APPROPRIATE

RECEIVED
MAY 0 8 2003
Technology Center 2100

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO
C.D		WO 96/24895 Hammond	08/15/1996	PCT/US	G06F	9/22		

OTHER DOCUMENTS *(Including Author, Title, Date, Pertinent Papers, Etc.)*

C.D			Kavi et al., A Performability Model for Soft Real-Time Systems, IEEE, Jan. 1994

EXAMINER [illegible]	**DATE CONSIDERED** 6/28/04

9427168.1

Notice of References Cited	Application/Control No. 09/322,443	Applicant(s)/Patent Under Reexamination REESE ET AL.	
	Examiner CHAMELI C. DAS	Art Unit 2122	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-6,721,941	04-2004	Morshed et al.	717/127
*	B	US-6,708,173	03-2004	Behr et al.	707/10
*	C	US-6,463,582	10-2002	Lethin et al.	717/158
*	D	US-6,256,775	07-2001	Flynn, William Thomas	717/127
*	E	US-6,351,646	02-2002	Jellema et al.	455/461
*	F	US-6,157,993	12-2000	Lewchuk, W. Kurt	711/213
*	G	US-6,553,431	04-2003	Yamamoto et al.	710/8
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	TITLE: The Profile Naming Service, author: Peterson, ACM, 1988.
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

The Profile Naming Service

LARRY L. PETERSON
University of Arizona

Profile is a *descriptive* naming service used to identify users and organizations. This paper presents a structural overview of Profile's three major components: a confedration of *attribute-based* name servers, a name space abstraction that unifies the name servers, and a user interface that integrates the name space with existing naming systems. Each name server is an independent authority that allows clients to describe users and organizations with a multiplicity of attributes; the name space abstraction is a client program that implements a discipline for searching a sequence of name servers; and the interface provides a tool with which users build customized commands. Experience with an implementation in the DARPA/NSF Internet demonstrates that Profile is a feasible and effective mechanism for naming users and organizations in a large internet.

Categories and Subject Descriptors: C.2.4 [**Computer Communication Networks**]: Distributed Systems—*distributed applications, distributed databases*; D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; H.2.4 [**Database Management**]: Systems—*distributed systems*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process, selection processes*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*information networks, question-answering (fact retrieval) systems*

General Terms: Design

Additional Key Words and Phrases: Attribute-based naming, white-pages service

1. INTRODUCTION

Profile is an *attribute-based* (*descriptive*) naming service for a large internet. It takes as an argument a set of attributes that describe a user or an organization and returns information about the resources associated with the entity identified by the attributes. Such a system is commonly said to provide a *white-pages* service. Profile is motivated by, and implemented in, the DARPA/NSF Internet [12, 21]. An important characteristic of such an internet is that it spans a collection of autonomous systems, each of which provides its own mechanism for naming resources. For example, a file name is meaningful in some file system, a host name is resolved by a host naming system, and mailbox addresses are understood in the mail system. Names defined in these systems are often difficult for users to discover for the first time and equally difficult to remember if used

This work was supported in part by NSF grant DCR-8609396.

Author's address: Department of Computer Science, University of Arizona, Tucson, AZ 85721.

infrequently. Profile is a supplemental naming service that augments existing naming systems; users consult it to learn the names of resources.

Profile is based on the architecture schematically depicted in Figure 1. The fundamental object supported the architecture is the *principal*: a user or organization that *sponsors* a collection of internet resources. For a principal to sponsor a resource implies that the principal owns, manages, or is somehow responsible for the resource.[1] For example, a principal might sponsor a mailbox, workstation, home directory, one or more printing devices, or a file system. Of the resources sponsored by principals, the one most important to Profile's architecture is the *name server*: a network server that takes as an argument a set of attributes describing one or more principals and returns information about the principals identified by the attributes. We say a name server is a *naming authority* for the set of principals whose names it is able to resolve.

For example, an internet site might constitute a principal that sponsors a name server. Such a name server would be a naming authority for the user, project, and role principals resident at the site. As another example, a mailing list, a users group, or a special interest group might sponsor a name server that is a naming authority for the group's members. One could also imagine an internet principal that sponsors a name server identifying those principals that sponsor a Profile name server. Finally, a user principal might sponsor a name server that is a naming authority for those principals whose resources the user frequently accesses. Such a name server is referred to as the user's *alias* name server.

Profile does not enforce a specific configuration, such as a hierarchy, on a collection of name servers. Instead, any principal can sponsor a name server and each such name server can be a naming authority for any set of principals. A name server that is an authority for principal *p* is said to be *linked* to a name server sponsored by principal *p* if, when asked to resolve a name for *p*, the former server returns the address of the latter server. For example, when asked to resolve the name of an organizational principal, the internet principal's name server might return the address of the organization's Profile name server. Intuitively, name servers are viewed as forming a loosely coupled confederation. Profile supports a *name space* abstraction that unifies this confederation by providing a mechanism for searching a sequence of name servers. The name space is not explicitly embedded in the individual name servers, but rather is implemented as a client program that takes advantage of links between servers.

Finally, Profile provides a user interface that integrates the Profile name space with the rest of the internet's naming systems. The interface provides a procedure-based tool with which users build customized commands.

This paper makes two major contributions. First, it gives a structural overview of the Profile naming service: Section 2 describes Profile name servers, Section 3 defines the name space abstraction, and Section 4 introduces the interface mechanism. Second, it discusses several issues associated with providing attribute-based naming systems and evaluates the techniques employed by Profile to address those issues. This discussion is presented in Section 5.

[1] Although we borrow the term "principal" from the computer security literature [23], we are interested in identifying principals in an effort to learn the names of their resources, not in how the system protects those resources.

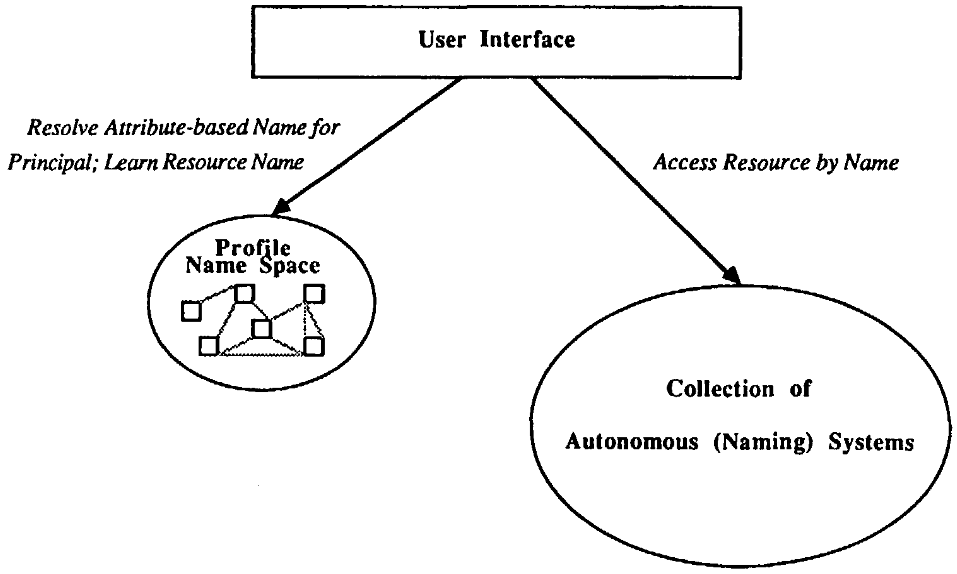
User Interface
Resolve Attribute-based Name for Principal; Learn Resource Name
Access Resource by Name
Profile Name Space
Collection of Autonomous (Naming) Systems

Fig. 1. Profile's architectures.

2. NAME SERVERS

A Profile name server is a naming authority for a set of principals, denoted **P**. A name server consists of three components: a database of *attributes* for the principals in **P**, a suite of *interpret* functions, and some database management tools. A client program running on behalf of a user contacts a name server and requests that a certain interpret function be applied to a given set of attributes, the interpret function computes the subset of principals in **P** identified by the attributes, and the name server returns the set of attributes it possesses for each identified principal. The client and name server communicate according to a specific protocol [4].

2.1 Attributes

An attribute is a syntactic entity of the form *tag=value* that denotes a property or characteristic of a principal. For example, 'mail=jxd@state.edu', 'name=John Doe', 'phone=621-1234' and 'login=jxd' are attributes that describe a user principal. A name server's database binds a set of attributes to each principal $p \in \mathbf{P}$.

$$\{attribute_1, attribute_2, \ldots\} \rightarrow p.$$

Each attribute bound to a principal is said to be *registered* for that principal. In the abstract, there also exists a possibly infinite set of attributes that *describe* each principal. A *name*, denoted $\mathbf{N} = \{attribute_1, attribute_2, \ldots\}$, is a finite set of attributes submitted by a user to identify one or more principals.

Intuitively, the attributes bound to a given principal correspond to both "external" identifiers for the principal—for example, personal name, phone number, postal mail address—and "internal" identifiers for the resources sponsored by the principal—electronic mail address, workstation name, login id.

Thus, while it might seem more natural to present a person's name to a Profile name server to learn his or her mailbox address, it is equally possible that a user might query a Profile name server with a mailbox address to learn a user's personal name. All attributes simply denote properties of principals.

Profile represents each attribute by an *expression* that defines a language over the alphanumeric characters, the blank character, and select special characters. Briefly, the operators that may be used to compose an attribute include alteration (|), optional substring ([ ]), zero or more character wildcard (*), and a single character wildcard (?); the full grammar for an attribute is given in the appendix. For example, the attribute 'John | [J [ohn]]Doe' denotes the set of strings 'John', 'John Doe', 'J Doe', and 'Doe'.

The attributes registered in the database must denote a finite set of strings and include the attribute tag as an optional prefix; e.g., '[name=] John | [John]]Doe' might be registered for principal John Doe. Attributes given in a name may take advantage of the full expressive power of the language, including the ability to denote an infinite number of strings. Typically, attributes given in names are simple expressions; e.g., 'John', '62?-1234', '*Doe'. Each attribute returned by a name server is a canonical form of the registered attribute; e.g., 'name=John Doe', 'mail=jxd@state.edu'. Based on its tag, a client may select an attribute from the set returned by a name server and use its value as a resource name that is meaningful in some other naming system.

A pair of attributes, for example one given in a name and one registered in the database, are considered equal if the intersection of the set of strings denoted by each is nonempty. Given this weak definition of attribute equality, a single attribute can be taken to represent different properties. For example, the attribute 'S*' might denote both a name and an address.

Profile defines a set of well-known attribute tags:

name	principal's name
address	principal's postal mail address
phone	principal's phone number
mail	principal's electronic mailbox address
login	user principal's login identifier
home	name of host on which principal's home directory resides
domain	principal's local domain name
profile	address of principal's attribute-based name server
handle	unique attribute by which principal can be identified in this name server

Additional attribute tags may also be included in a name server. Note that not all attribute tags are used for all principals. For example, the **login** tag is only meaningful for a principal that corresponds to a user, while the **profile** tag is only meaningful for a principal that sponsors a Profile name server. Also, each principal need not have the full set of attribute tags bound to it in the name server. For example, a user principal might have a postal mail address but no phone number in the database. Finally, multiple attributes with the same tag may be bound to a single principal, denoting multiple instances of the same resource; e.g., multiple mailboxes, replicated Profile name servers.

2.2 Interpret Functions

Profile supports a family of interpret functions, each of which computes the set of principals identified by a given name according to the attributes registered in the name server's database. The interpret functions are described in intuitive terms for the purpose of this paper. A companion paper develops a theoretical framework for reasoning about naming systems as specialized inference mechanisms [5]. The theory is used to characterize a class of *descriptive* naming systems that includes Profile. The class is less restrictive than conventional naming systems, but more specialized than general databases.

Mapping sets of attributes onto principals is nontrivial for four reasons. First, one attribute may describe many principals. Second, some attributes in the name server's database may not correctly describe the principals for which they are registered. Third, not all attributes that describe a given principal are registered for that principal. Finally, users may submit attributes that do not describe the principal they were intended to describe. We say that an attribute is *ambiguous* if it describes more than one principal and an attribute is *invalid* if it does not describe a particular principal but is either registered for the principal or given in a name for the principal. Profile's interpret functions are based on the following definitions.

First, the set of attributes registered in the name server's database is partitioned into *complete* and *incomplete* subsets: an attribute registered for principal p is complete if and only if (i) it describes p, and (ii) it is registered for *all* principals in **P** it describes. All other attributes are incomplete. Note that the second condition for completeness corresponds to the *closed world* assumption in logic databases [22]. Clearly, an interpret function cannot reliably use incomplete attributes. One reason, of course, is that incomplete attributes may be invalid. A second, and in practice, more likely reason is that an incomplete attribute may be registered for one principal it describes, but not another. To see the significance of this second point, consider attributes a and b, both of which describe principals p and q. Suppose a is registered for both principals but b is registered only for p; that is, a is a complete attribute and b is an incomplete attribute. If a user names a principal with the attributes a and b, then it is intuitively wrong to not return q simply because attribute b is not registered for it; name $\{a, b\}$ describes both principals equally well.

Second, an attribute in name **N** is considered a *candidate* if and only if the attribute is registered for some principal in **P**. For example, suppose the name server's database contains the following two bindings

$$\{a, b, c\} \rightarrow p$$
$$\{a, b, d\} \rightarrow q.$$

Name $\{a, c, d, e\}$ contains the candidate attributes a, c, and d. Intuitively, noncandidate attributes are either not registered for the principal they describe or they are invalid. Furthermore, a set of candidate attributes is partitioned into two disjoint subsets, called the *complete candidate name* and the *incomplete*

Fig. 2. Definition of *interpret*$_1$.

```
interpret₁(N)
{
        return(cp(N) ∪ ip(N));
}
```

candidate name, based on whether each attribute is in the complete or incomplete partition of registered attributes, respectively.

Third, for any two sets of attributes—e.g., a complete candidate name and the set of complete attributes registered for a principal—the first set is said to *exactly match* the second set if and only if the former set is totally contained in the latter set, and *partially match* if and only if the intersection of the two sets is nonempty. For example, the candidate name $\{a, b, d\}$ exactly matches the set of registered attributes $\{a, b, c, d\}$, but the candidate name $\{a, c, e\}$ only partially matches $\{a, b, c, d\}$. In other words, Profile defines two standards for determining the equality of a set of attributes in a name and a set of attributes registered for a principal—exact match and partial match—both of which are weaker than set equality.

Thus, for name **N** and a set of bindings for principals in **P**, with the attributes registered for each principal partitioned *a priori* into complete and incomplete subsets, we can define the following four sets:

ce(N) ≡ the subset of principals in **P** whose complete attributes are exactly matched by the complete candidate name;

ie(N) ≡ the subset of principals in **P** whose incomplete attributes are exactly matched by the incomplete candidate name;

cp(N) ≡ the subset of principals in **P** whose complete attributes are partially matched by the complete candidate name;

ip(N) ≡ the subset of principals in **P** whose incomplete attributes are partially matched by the incomplete candidate name;

Note that the set of principals exactly matched by a complete (incomplete) candidate name is a subset of the partially matched principals; i.e., $\mathbf{ce(N)} \subseteq \mathbf{cp(N)}$ and $\mathbf{ie(N)} \subseteq \mathbf{ip(N)}$ for all **N**. Also, if **ce(N)** is empty but **cp(N)** is not, then the user must have submitted invalid attributes for the intended principal.

Profile provides four interpret functions, each of which defines an increasingly restrictive (discriminating) interpretation of what principals are named by a set of attributes. The least restrictive, called *interpret*$_1$, and given in Figure 2, computes the disjunction of the attributes contained in the name.

Loosely speaking, the second and third functions compute the conjunction of the attributes in the given name, thereby allowing the user to identify some particular principal. *Interpret*$_2$, as illustrated in Figure 3, gives priority to complete candidate names over incomplete candidate names and exact matches over partial matches. For a name that contains a nonempty complete candidate name and no invalid attributes, the set of principals returned by *interpret*$_2$ is guaranteed, by definition, to include the intended principal.

Interpret$_3$, as given in Figure 4, also prefers complete attributes to incomplete attributes and exact matches to partial matches. *Interpret*$_3$ is more restrictive

```
interpret₂(N)
{
        if ( |ce(N)| > 0)
                return (ce(N));
        if ( |cp(N)| > 0)
                return (cp(N));
        if ( |ie(N)| > 0)
                return (ie(N));
        return (ip(N));
}
```

Fig. 3. Definition of *interpret*$_2$.

```
interpret₃(N)
{
        if ( |ce(N)| = 1)
                return (ce(N));
        if ( |ce(N) ∩ ie(N)| > 0)
                return (ce(N) ∩ ie(N));
        if ( |ce(N) ∩ ip(N)| > 0)
                return (ce(N) ∩ ip(N));
        if ( |ce(N)| > 0)
                return (ce(N));
        if ( |cp(N) ∩ ie(N)| > 0)
                return (cp(N) ∩ ie(N));
        if ( |cp(N) ∩ ip(N)| > 0)
                return (cp(N) ∩ ip(N));
        if ( |cp(N)| > 0)
                return (cp(N));
        if ( |ie(N)| > 0)
                return (ie(N));
        return (ip(N));
}
```

Fig. 4. Definition of *interpret*$_3$.

than *interpret*$_2$, however, because it uses incomplete attributes to narrow the set of returned principals. If N contains no incomplete attributes, then *interpret*$_3$ returns exactly the same set of principals as *interpret*$_2$. It cannot, however, guarantee that the intended principal is returned if N contains an incomplete attribute.

Finally, *interpret*$_4$ given in Figure 5 uses **ce(N)** to match at most one principal. Intuitively, *interpret*$_4$ implements the lookup function provided in many conventional directories; it fails when the name is ambiguous.

```
interpret4(N)
{
        if ( |ce(N)| = 1)
                return (ce(N));
        return (∅);
}
```

Fig. 5. Definition of *interpret*$_4$.

To see the distinction between the four interpret functions, consider a database that contains the bindings

{John, Smith, systems} $\rightarrow$ *js*
{John, Doe} $\rightarrow$ *jd*
{Mary, Smith, theory} $\rightarrow$ *ms*

where attributes 'John', 'Mary', 'Smith', and 'Doe' are complete, attributes 'systems' and 'theory' are incomplete, and *js*, *jd*, and *ms* denote three different principals. (The policy for determining which attributes are complete and which are incomplete is discussed in Section 2.3.) When applied to the name {Harry, Smith}, the first three interpret functions return *js* and *ms* and *interpret*$_4$ returns the empty set; when applied to the name {Mary, Smith}, *interpret*$_1$ returns *js* and *ms*, while the last three interpret functions return only *ms*; and when applied to the name {John, systems}, *interpret*$_2$ returns *js* and *jd*, *interpret*$_3$ returns only *js*, and *interpret*$_4$ returns the empty set.

A final note: If there exists a subset of complete attributes registered for a particular principal that does not exactly match the complete attributes registered for some other principal—i.e., the principal's name is not "hopelessly" ambiguous—then it is possible to construct a name such that *interpret*$_2$, *interpret*$_3$, and *interpret*$_4$ return only that principal. To guarantee this property, as well as to provide an attribute-based name for each principal that is likely to match exactly the same principal for multiple queries of a name server, each Profile name server defines at least one unique attribute for each principal. This unique attribute has the tag **handle**.

2.3 Database

In most ways, a Profile name server database is managed like any other name server database [3]. This section briefly considers those database issues that are unique to Profile. A more detailed discussion of the implementation can be found in [4].

From an operational perspective, a name server's database exists in two forms. A text version contains a set of complete and incomplete attributes for each principal in **P**. The text version is created and updated by users. An internal version of the database is used by the interpret functions. The internal version is compiled from the text version by translating the set of registered attributes into a finite state automata. The name server is restarted periodically so that the internal version of the database reflects changes to the text version.

The text version of the database is divided into protected and unprotected partitions. The protected partition consists of the complete attributes for each principal in **P**. The protected partition can only be updated by a privileged administrator. It is the responsibility of this administrator to ensure that the complete attributes registered in the database satisfy the definition of completeness; i.e., that they are correct and that an attribute registered for one principal is registered for all principals it describes. Satisfying the first part of this definition assumes the correctness of information given the privileged administrator by individual principals; an assumption we find acceptable. A sufficient, but not necessary, condition for guaranteeing the second part of this definition is to register all attributes with a given tag; e.g., all phone numbers, all personal names.

The exact way in which the protected partition of the text version is maintained depends on the set of principals for which the name server is an authority. Consider, for example, a Profile name server sponsored by an internet site for which **P** is the set of individuals, roles, and projects within the site. Complete attributes for these principals can be automatically generated from existing administrative data. The prototype implementation in UNIX®, for example, uses an **awk** script [1] to generate the set of attributes

```
{[login=]jxd, [name=]John|[J[ohn]]Doe,
[office=]320, [phone=][(621|×1|×1)[−| ]]1234, [mail=]jxd@state.edu}
```

for user John Doe from the mail alias file **/usr/lib/aliases** and the **/etc/passwd** entry

```
jxd:r4zqXMNgxssyM:75:20:John Doe,302UCC,1234:/usr1/jxd:/bin/csh
```

As another example, a special interest group or a users group might generate a text database from its membership list. Note that, in both cases, the attribute generation mechanism is reexecuted periodically (i.e., daily) to incorporate changes in the administrative data. Section 3 describes an alternative mechanism used by individuals to maintain alias name servers.

The unprotected partition of the database contains incomplete attributes for each principal. Principals are allowed to modify the unprotected information bound to them in the text version of the database. For example, a user could enter additional attributes that might help others identify the user (e.g., nicknames, areas of interest, current projects), as well as attributes that correspond to additional resources associated with the principal (e.g., printer, workstation). Attributes registered in this manner must be treated as incomplete. This is because there is no guarantee that all users will register a particular attribute. There is also no guarantee that the attributes are correct; i.e., one user might masquerade as another by entering the other user's attributes. Protection mechanisms available in the local system are used to ensure that a given user has the appropriate permission to modify the information bound to a given principal; e.g., a user can modify his or her own binding.

Note that even though an organization supports a single name server that is an authority for all the user principals within the organization, the administration of the name server's text database might be distributed over a hierarchy. For example, each department in a university might maintain a text database for its

® UNIX is a trademark of Bell Laboratories.

own users, with the resulting set of text databases merged to generate a single internal database for the university's name server. Also, a principal is free to replicate name servers. That is, a principal might implement a set of name servers on distinct hosts, each of which is an authority for the same set of principals and each of which is initialized from the same text database. Finally, allowing users to modify the unprotected partition of the database is best suited for a site-based name server where local users have more convenient access to the database. A name server sponsored by a special interest group, for example, might implement only the protected partition and offer a name server supporting only complete attributes.

3. NAME SPACE

A name space abstraction unifies the confederation of name servers. It provides an extended syntax for specifying attribute-based names, called a *profile name*, a discipline for contacting a multiplicity of name servers, and a mechanism for maintaining an alias name server. The name space is implemented as a client program that takes advantage of profile attributes returned by one name server that give addresses of other name servers. Note that allowing any name server to have a link to (capability for) any other name server, thereby resulting in a nonhierarchical confederation name space, is not new to Profile. Such name space organizations have been used successfully in other computing systems as well [2, 15, 25]. What makes searching the name space more difficult in Profile is that the name being resolved at any given name server may yield multiple links (capabilities).

The name space supports two operations, *resolve* and *bind. Resolve* takes as an argument a profile name and returns the set of attributes associated with each principal identified by the name. Three parameters defined in a client's environment control the operation of *resolve*: a *search path* specifies the sequence of name servers to be contacted to resolve the name; a *default interpret function* defines the interpret function that is to be applied at each name server; and a *search strategy* specifies one of four possible disciplines for *resolve*. The elements of the search path are name server addresses known a priori by the client, the interpret function is given by a string that denotes the function, and the search strategy is expressed as a product of two orthogonal options: (i) find the *first* or find *all* results, and (ii) follow links under *explicit* or *implicit* control.

In its simplest form, the profile name taken as an argument by *resolve* is a string consisting of one or more attributes separated by delimiters; e.g., 'John Doe, 621-1234, State University'. We call this an *attribute-based name*. In addition, a profile name may contain two or more attribute-based names separated by an indirection operator '@' and an alternation operator '|'; e.g., 'John Doe, 621-1234 @ State University' and 'John Doe, 621-1234 @ (State University | Sigcomm)'. Also, a string denoting an interpret function may be explicitly applied to an attribute-based name; e.g., 'John Doe, 621-1234 @ interpret1(State University)'. Finally, a profile name may contain multiple '@' and '|' operators, where '|' has precedence over '@', '|' is left associative, and '@' is right associative.

To illustrate how *resolve* operates on the four example profile names given in the previous paragraph, consider the following parameter settings. Suppose a user knows that the local site's name server is located on the host named 'megaron', the interpret principal sponsors a name server that resides on the host named 'sri-nic.arpa', and a special interest group to which the user belongs sponsors a name server on host 'venera.isi.edu'.[2] The user might specify the search path

⟨alias, megaron, sri-nic.arpa, venera.isi.edu⟩,

where 'alias' is a label for the user's alias name server. Also assume the user defines $interpret_2$ as the default interpret function and specifies a "find the first answer under explicit control" search strategy.

Given this environment, the first profile name is interpreted at each name server in the search path, in order, until one of the name servers yields a nonempty result. In the case of the second profile name, the attribute-based name to the left of the '@' is interpreted at the name server sponsored by each principal identified by the attribute-based name to the right of the '@'. The later name is interpreted according to the search path. Should the rightmost attribute-based name yield a **profile** attribute for more than one principal, then *resolve* contacts the name server sponsored by each such principal to interpret the leftmost name and the union of the results from each name server is returned. The third example name causes the leftmost attribute-based name to be resolved at the name server sponsored by the principal(s) identified by 'State University' and the principal(s) identified by 'Sigcomm'; both attribute-based names right of the '@' are interpreted independently according to the search path. The final profile name uses $interpret_1$ to resolve the rightmost name and the default interpret function to resolve the leftmost name.

Now consider how alternative parameter settings vary the resolution process. First, by setting the search strategy to look for all results rather than the first result, a profile name is interpreted at all name servers identified in the search path and the union of results returned from each server is returned to the client. An alternative proposed by Sollins that generalizes the "first" and "all" strategies is to let the search path be a partial order [25].

Second, by setting the search strategy to implicitly follow links, *resolve* automatically contacts any name servers identified in a result yielded by a query of another name server; i.e., both the '@' and '|' operators are unnecessary. In other words, using this search strategy to resolve the name 'John Doe, 621-1234, State University, Sigcomm' usually yields the same results as the third example given above. It may, however, yield unexpected results. For example, if the search strategy is also set to look for the first result and John Doe is the name of a local principal, then *resolve* will return information about the local John Doe and not one at State University or at Sigcomm. An even more subtle possibility is that the user intends for attributes 'State University' and 'Sigcomm' to identify a principal at the internet principal's name server and for 'John Doe' and '621-1234' to yield a result at each of those principals' name servers, while in fact

[2] This example is for illustrative purposes only.

'John Doe' yields a result at the internet name server (perhaps it is the name of a company) and 'State University' identifies a principal known in John Doe's name server (perhaps John Doe the company has an affiliation with State University). It is also possible that following links under implicit control results in a loop; i.e., one name server being contacted multiple times to resolve the same name. Such loops are detected and broken by the client program that controls the search.

In addition to resolving profile names, the name space *bind* operation provides an interface to the user's alias name server. It takes as arguments a set of attributes and adds an entry containing the attributes to the user's alias name server. Typically, the set of attributes are given by an "alias" attribute and the result of calling the *resolve* operation. For example,

bind('john', *resolve*('John Doe, routing, Sigcomm'))

adds an entry containing 'john' and the attributes returned by *resolve* to the user's alias name server.

We conclude this section with several comments. First, while it is tempting to think of the internet principal's name server as the root of a hierarchy and a name of the form 'John Doe @ State University' as a path name, it is more accurate to view the '@' operator as specifying one level of indirection. For example, if the local site's name server identifies the site principal named 'State University' and the local name server is found in the search path before the internet principal's name server, then the name 'John Doe @ State University' is resolved without consulting the internet principal's name server.

Second, control over the search process is centered at the client. This is in keeping with the philosophy that the name space is an "end-to-end" function defined by the client; name servers never contact each other on behalf of a client, and in fact, they have no knowledge of the existence of a name space. This design has two important consequences. The first is that alternative name space abstractions can be defined, each of which supports a different syntax and more or less restrictive searching machinery. The second is that it is easy for the client to terminate a search before it has completed. This might happen, for example, if the user is satisfied with the first answer returned. As pointed out earlier, such termination is essential when the client program detects a loop.

Finally, the user can step through the confederation of name servers in a piecemeal fashion, rather than submit one complex profile name. For example, a user might resolve the simple name 'State University', extract the address of that principal's name server from the result, and then contact that name server with one or more other profile names. As another example, a user might submit the name 'John Doe @ Sigcomm', learn that John Doe is at State University, and then submit the name 'John Doe @ State University' in the hope of learning more attributes for John Doe from State University's name server than he or she was able to learn from Sigcomm's name server.

4. USER INTERFACE

A user interface provides a mechanism through which users interact with Profile and invoke base commands in the underlying computing system. In its simplest form, a user interface to Profile might apply *resolve* to a given profile name and display the resulting attributes to the user, leaving the user responsible for composing a resource name and applying a base command to the name. A more complicated user interface might fully integrate the Profile name space with the existing naming systems, thereby allowing the user to apply base commands to profile names directly. Such an interface hides the underlying systems behind the principal abstraction. Between these two extremes, the user might implement a collection of customized commands, each of which augments some base command with a routine that takes a profile name as an argument, invokes the *resolve* operation to retrieve the set of attributes bound to the principal identified by the name, and translates the attributes into a resource name understood by the base command. For example, a customized **mail** command might allow a user to address mail to a principal rather than a mailbox; e.g.,

mail "John Doe, Computer Science, State University"

Recognizing the common thread that runs through such customized commands, Profile provides a procedure-based translation mechanism that is used to construct such commands. We refer to customized commands built using the translation mechanism as *profile commands.* This section gives an operational overview of the translation mechanism, and demonstrates how the mechanism has been incorporated into the UNIX **csh** shell [8].

4.1 Translation Mechanism

A profile command *specification* is an ordered list of *conditional templates*, each of which is a triple of the form

$$\langle \mathit{condition},\ \mathit{cmd\_name},\ \mathit{rewrite\_rule} \rangle$$

A *condition* is a Boolean function that takes one or more attributes as arguments. For example, a condition that determines if a principal sponsors a Profile name server is a function of the **profile** attribute, a condition that determines if a principal is a user is a function of the **login** attribute, and a condition that determines if a principal's resources are contained in the local system is a function of the **domain** attribute. A *cmd_name* is the name of a base command that is to be applied to a resource. It must be understood in the local computing system. A *rewrite_rule* is a function that takes one or more attributes as arguments and returns a resource name.

Suppose, for example, that a user defines a specification for a profile command called **copy** that copies files to and from a principal's home directory, and a specification for a profile command called **mail** that sends mail to a principal. Borrowing syntax from the C programming language [9][3], Figure 6 gives the specifications for the two profile commands, where each condition is given by a C procedure similar to the one presented in Figure 7.

The translation mechanism takes a profile command specification and a set of attributes as arguments and returns the name of a base command and the name of a resource that the base command takes as an argument. Operationally, the translation mechanism evaluates the condition associated with each template in the specification according to the attributes, where the first template for which

[3] The string contained between double quotes controls C's formatted output.

```
copy  ≡  ⟨⟨ local_user(login,domain), cp, ("/r/%3s/usr/%s",home,login)⟩;
          ⟨ local_sys(login,domain), cp, ("/r/meg/usr/ftp")⟩,
          ⟨ remote_user(login,domain), rcopy, ("%s:%s.%s:",login,home,domain)⟩,
          ⟨ remote_sys(login,domain), rcopy, ("anonymous:%s.%s:",home,domain)⟩⟩

mail  ≡  ⟨⟨ local_user(login,domain), mail, ("%s",login)⟩;
          ⟨ def(login), mail, ("%s",mail)⟩;
          ⟨ !def(login), mail, ("root@%s",domain)⟩⟩
```

Fig. 6. Example Profile command specifications.

```
local_user(login, domain)
char *login, *domain;
{
        if (strlen(login)>0 && strcmp(domain,"state.edu")!=0)
                return(TRUE)
        else
                return(FALSE)
}
```

Fig. 7. Example condition.

the condition is satisfied yields a command name and a rewrite rule. The translation mechanism returns the command name and the string produced by applying the rewrite rule to the attributes. For example, when given the specification for the **copy** profile command and the set of attributes

{login=jxd, name=John Doe, phone=621-1234, mail=jxd@state.edu, domain=state.edu, home=megaron}

the translation mechanism returns the command name 'cp' and the resource name '/r/meg/usr/jxd'.

4.2 Profile Shell

The Profile shell, **pcsh**, is an extended version of the UNIX **csh** that supports the specification and invocation of profile commands. Briefly, **csh** provides a command interpreter that accepts input from the user and invokes commands in the underlying UNIX system. Each line of input is processed by the five phases given in Figure 8. In addition, **csh** defines an environment that consists of a set of shell variables (e.g., a search path variable used to locate binary files that implement commands) and specifications for various phases of the interpreter (e.g., aliases used in the rewriting of commands in phase two). **Pcsh** extends **csh** as follows.

First, the lexical analysis phase is modified to recognize arguments that contain, either totally or in part, profile names. Such names are syntactically distinguished with special characters; e.g., double quotes. Second, the environment is extended to include the three name space parameters defined in Section 3, as well as a

(1) Lexical Analysis
(2) Aliases Substitution
(3) Parse
(4) File Name Completion
(5) Execute

Fig. 8. Five phases of **csh**.

command registry that binds profile command names to their respective specifications. Third, a new profile phase is added after the parsing phase and before the file name completion phase. The profile phase invokes the *resolve* operation to retrieve the set of attributes associated with any profile names tagged by the lexical analysis phase; retrieves the specification bound to the command name from the command registry; applies the translation mechanism to the attributes and specification; and substitutes the resulting command name and resource name for the original command and arguments.

For example, given the set of attributes bound to John Doe and the profile command specifications for **copy** and **mail**, the input line

copy "John Doe"/project/src/x.c y.c

is rewritten as

cp/r/meg/usr/jxd/project/src/x.c y.c

if John Doe is a local user, and the input line

mail "John Doe @ State University"

is rewritten as

mail jxd@state.edu

if John Doe is a remote user. In the former example, if the profile name 'John Doe' had identified a remote user rather than a local one, then rcopy would have been applied to the arguments 'jxd:megaron.state.edu:src/x.c y.c', where rcopy is an interface to FTP [20] that takes the leftmost component of the argument (jxd) to be a login identifier, the intermediate component of the argument (megaron.state.edu) to be a host name, and the remainder of the name (project/src/x.c) to be a file name. In the case of a remote system, rcopy would have established an FTP connection to the system's guest account.

A final note: If a command is applied to an argument that contains a profile name, but no specification is bound to the command, then the original argument is passed directly to the command. Passing uninterpreted profile names to commands allows more complex integration of Profile with the application. For example,

mail "John Doe"@state.edu

causes a message to be forwarded to the host identified with the domain name 'state.edu', where the mail program running on that host invokes a local Profile name server to interpret 'John Doe'. Passing profile names directly to commands also facilitates a **profile_lookup** command that provides a direct interface to the *resolve* and *bind* operations provided by the name space abstraction.

5. EXPERIENCE AND EVALUATION

Profile has evolved to its present form over a period of two years. It currently runs on a testbed of three sites and has been used by several dozen users. This section evaluates the successes and limitations of Profile and reports how our experiences with Profile reflect on several issues related to attribute-based naming.

5.1 Name Servers

Profile name servers allow multiple attributes to be registered for each principal, support a flexible representation of attributes, and provide a suite of interpret functions that can be applied to attribute-based names. We have considerable experience with individual Profile name servers, including extensive testing during the development phase, comments from our user community, and the intuition developed while exploring the theoretical issues. Based on this experience, we comment on the following issues.

5.1.1 *Attributes.* Although Profile attributes are tagged, clients are not required to include the tag when they query a Profile name server. The reason is that tags are strings concatenated to attributes to enforce a uniqueness property [6], that is, to make attributes that denote different properties syntactically unique. While pathological examples exist, our experience strongly suggests that attributes denoting different properties are already unique, and therefore tags usually need not be included. We believe this to be the case because different types of attributes are inherently represented by different expressions—personal names do not look like street addresses, which do not look like phone numbers, and so on. Another way of looking at this is to say that an attribute's tag can be determined from the syntax of its value. Giving an attribute's tag does improve the accuracy of the interpret functions, however, when the attribute's value is ambiguous. For example, if a user knows that a principal's name begins with 'S' but has no hint as to the correct spelling, an attribute of the form 'name=S*' is much more likely to produce a meaningful result that is the attribute 'S*'.

We had two objectives in mind when choosing to represent attributes by expressions. First, we wanted a flexible mechanism for comparing named attributes against registered attributes, thereby allowing users a wide latitude in specifying attributes. Second, we wanted to be able to build simple tools for generating attributes from available administrative data. Although expressions have proven adequate in both regards, it is clear that how attributes are generated, represented, and compared is the aspect of Profile that requires the most fine tuning. An example problem is the treatment of common nicknames, such as Doug for Douglas, Rick or Dick for Richard, and so on. One promising idea with which we have experimented is a mechanism that determines whether or not two strings (e.g., 'Doug' and 'Douglas') represent the same attribute by computing the ratio of the number of characters by which the two strings differ to the total number of characters in the strings [14]. Although this technique accommodates simple nicknames, it is not able to detect that 'Peggy' is a nickname for 'Margaret'. A more general solution is to write an expert system that generates nicknames to register in the database. Also, users can directly enter additional nicknames as incomplete attributes. Finally, other possibilities that we have not yet explored involve the use of spelling correctors and phonetic spelling programs.

5.1.2 *Interpret Functions.* We have demonstrated experimentally that Profile's interpret functions are an effective tool for identifying principals. Our experience shows, for example, that *interpret*$_1$ is useful when the user's intent is to identify as wide a set of principals as possible; *interpret*$_2$ is appropriate when the user wants to ensure that a particular principal is returned, possibly at the expense of multiple principals being returned; *interpret*$_3$ offers a useful heuristic for pruning extra principals from the result; and *interpret*$_4$ is appropriate when the user wants a unique match. *Interpret*$_2$ is the function of choice for identifying a single principal.

Although difficult to quantify, Profile's interpret functions appear to offer a richer mechanism for computing the principals identified by an attribute-based name than do similar white-pages naming services; e.g., X.500 [7], NICNAME [19], and the CSNET name server [26]. This is because Profile considers both the exactness of the match and the potential incompleteness and inaccuracies of the database. Moreover, being able to tune the behavior of a name server by selecting the interpret function is a strength of Profile. In regard to this latter point, it is not clear that a more powerful mechanism that supports arbitrary queries—e.g., a general relational database or the filter mechanism provided by X.500—is warranted. To illustrate this point, consider that an interpret function can be viewed as a program that takes a set of attributes as an argument, constructs a specific database query, and solves it. Each of Profile's interpret functions is designed to a construct a useful class of queries. To achieve the same effect with a general database would require the user either to compose an extremely complex query, or to use a query language mechanism that, loosely speaking, implements the interpret functions.

5.1.3 *Registering Attributes.* Consistent with experience from similar efforts, getting users to register information is the most difficult aspect of providing a white-pages naming service. We note, however, that by providing Profile name servers at the local-level rather than the internet-level, and by further distributing the administration of a single name server over a hierarchy of text databases, we have met with some success in this regard.

It is also the case that a major success of our design is that we have been able to generate name server databases from existing administrative data, thereby freeing the name server from depending on individual users. Moreover, we are able to re-execute the attribute generation mechanism periodically to keep the database up-to-date. Finally, the possibility of one principal masquerading as another by registering attributes for himself or herself that really identify someone else is not a problem. The reason for this is that individual principals are only permitted to enter incomplete attributes and complete attributes are preferred to incomplete attributes by the interpret functions.

5.1.4 *Performance.* An internal version of a database (i.e., one containing attributes that have been expanded) for a minimally configured Profile name server (i.e., one containing only complete attributes) that is an authority for 3,050 principals occupies approximately 30K bytes of memory. Experiments with the name server show that *interpret*$_3$, the most complex interpret function, computes the set of principals that match a typical attribute-based name in 55 msec on a VAX 8600. These results suggest that the current implementation of a name server should be able to accommodate on the order of 10^4 principals. A single name server should therefore be sufficient for a large university or company. We are currently experimenting with indexing techniques that we expect to allow a name server to handle on the order of 10^5 principals efficiently.

5.1.5 *Protection.* Although not currently implemented in Profile, we understand that some protection mechanism, e.g., an access control list, is essential [23]. Such a mechanism would allow the name server's sponsor to define different client classes and to permit clients in each class to invoke some subset of interpret functions (thereby controlling the precision with which a client must describe a principal before the name server returns information about the principal) and receive some subset of attributes for each identified principal (thereby controlling how much information the name server releases about each principal identified by the client). For example, an organization's name server might allow local users to invoke all the interpret functions and receive all attributes registered for any identified principals, but permit remote users to invoke only *interpret*$_2$ and *interpret*$_4$ and receive only the **name** and **mail** attributes for any identified principals. Such restrictions can be applied uniformly across the entire database, as well as on an entry by entry basis; e.g., one principal may restrict access to the **phone** attribute and another may not.

In addition to restricting the interpret functions that may be applied and the attributes that may be returned, a name server sponsor can tune the protection policy in two additional ways. First, the precision of attributes registered in the database may be varied; e.g., the attribute 'John Doe' might be registered as a person's name rather than '[J[ohn]]Doe'. The more precise the representation of attributes, the more exact the information presented by the client to identify a principal. Second, the name server can restrict the form of attributes it accepts from clients. For example, it can choose to not resolve attributes containing the '*' operator.

5.1.6 *Relocation Transparency.* Profile does not attempt to make the relocation of principals—that is, changes to the attribute values bound to them—transparent. Our experience suggests that if the relocation of principals is to be made transparent, then it should be done by a lower level naming system. For example, to facilitate sending mail to a user principal in a relocation transparent way, one of the attributes returned for a user principal could be an internet-wide unique identifier that is in turn mapped into the user's current mailbox address by some independent naming service.

Profile does support a unique **handle** attribute, however. A principal's handle is never reassigned, even if the name server is no longer an authority for the principal. Once a client possesses a principal's handle, for example as a result of an initial query of a name server, it can reuse the handle as an attribute in subsequent queries of the name server to retrieve the principal's current attributes. Thus, handles are useful for refreshing the attributes bound to a principal in a user's alias name server.

5.1.7 *Naming Arbitrary Objects.* Profile was originally designed to name a single type of object: principals. Clients indirectly learn the names of other resources by giving attributes that identify their sponsor. It is also desirable to allow clients to name resources by giving other attributes that describe the resource. This would allow a client to distinguish among a set of similar resources sponsored by the same principal. For example, a client might want to identify a local printer by submitting the attributes 'architecture=postscript', 'pagesperminute>8', and 'load=min'. We have successfully extended a Profile name server to allow clients to identify arbitrary objects, including computational objects such as printers, processors, and databases, with any attribute that meaningfully describes the object [4, 17].

While name servers that resolve attribute-based names for computational objects are often referred to as *yellow-page* servers, our experience is that yellow-page and white-page servers are fundamentally the same. In particular, we are able to support both within the framework of the same architecture simply by including a new interpret function, defining alternative attribute representations (e.g., integers, reals, and lists in addition to expressions) and supporting indefinite attributes (i.e., attributes that include relational operators like < and >). It is also worth noting that a technique similar to the attribute generation mechanism, used to register attributes for principals, is used to register frequently changing attributes for computational objects; e.g., a processor's load. Instead of the conventional model of registering and explicitly updating the database with static values, the technique involves registering functions that compute attribute values dynamically and on demand. Finally, adding arbitrary objects to a Profile name server results in a significant number of cross-references between objects, thereby allowing one to give an attribute-based name for a user to learn the name of the user's workstation, as well as to give an attribute-based name for a workstation to learn the name of the user that owns the workstation.

Adding different objects to a name server introduces one new problem: attribute values with the same tag but registered for different types of objects are quite often indistinguishable. For example, submitting the attribute-based name 'name=john' might yield both a workstation and a user. We have, therefore, defined a new attribute tag, denoted **is_a**, that identifies the type of the object. However, we treat an object's **is_a** attribute as simply another attribute that describes the object: the **is_a** attribute need not be given to identify the object, but doing so improves the chances of receiving the expected answer. Note that the problem of the same attribute being registered for different types of objects is different from the point made in Section 5.1.1 that attributes for the same type of object need not be tagged because their values are syntactically unique. It is the case, however, that the argument made in 5.1.1 does not apply as well to computational objects as it does to principals. The reason is that different attributes for computational objects are often given by small integers.

5.2 Name Space

Profile adopts a "library reference collection" model for its name space: name servers are independent naming authorities arranged in a loose confederation, the name space abstraction is implemented at the client, and a flexible search strategy is used to traverse a set of name servers. The testbed Profile name space contains a collection of alias name servers and five organizational name servers—an internet name server, three site name servers, and a name server consisting of the members of the ICON user's group. It is clear that the size of the testbed is too limited to support a thorough analysis of the name space. In particular, it is difficult to estimate the network load Profile would generate, and as a consequence of that load, identify what restrictions should be placed on the search strategies. On the other hand; we have demonstrated the workability of the name space, and we are able to offer the following preliminary evaluation regarding the appropriateness of the design.

5.2.1 *Performance.* As indicated in Section 5.1.4, a single name server is capable of serving large organizations on the order of universities and companies. As a consequence, it is reasonable to expect a site-based collection of name servers and a single level of indirection to be sufficient for the current configuration of the DARPA/NSF Internet. In such an environment, the following performance factors are important: the alias name server can be accessed by a procedure call; the local site name server can be queried in under 100 msec; it can take several seconds to query a name server across the DARPA/NSF Internet; the '@' operator often implies that two remote servers need to be queried; and each server identified in the search path may need to be queried. The last point is particularly critical if the name contains an error or identifies an unknown principal.

Our experience shows that the performance of *resolve* can be made satisfactory, that is, the cost of accessing a remote name server is incurred only when necessary, when the "find the first answer under explicit control" search strategy is used. To do this, the user must exploit the alias name server and take care to define a minimal search path. That is, the search path should include only the alias and local name server unless the user wants to conduct a global search. Although not implemented in Profile, our experience suggests that the search path be partitioned into "local" and "global" parts and that the user be asked to confirm use of the global part.

5.2.2 *Separation of Functionality.* One of the most important design philosophies behind Profile is that an internet's attribute-based naming service should be separate from its *universal* name space, and that the design of one should not be compromised for the sake of the other. Internet universal naming systems are typified by the DARPA/NSF Internet's domain naming system [13]; similar naming systems include Lampson's global naming service [10], Lantz, Edighoffer, and Hitson's UDS system [11], and the Xerox Clearinghouse [16].

Our design advocates this separation because the primary goal of a universal naming system is to support absolute names that can be resolved efficiently. For example, the domain system must map a host name into the same internet address for all clients, and this mapping must be performed every time a client establishes a connection to a host. In contrast, Profile's objective is to provide a flexible naming service that maximizes the possibility of the user finding a useful result. It does this at the expense of preserving absolute names (users share "descriptions" of principals rather than precise profile names) and at the expense of efficiency (users only engage Profile to *learn* resource names, not every time the resource is accessed).

To realize its goals, universal naming systems generally restrict the structure of the name space to a hierarchy and support a simple search strategy in which a name denotes a path through the hierarchy. In contrast, Profile allows an arbitrary configuration of name servers and offers a richer search strategy. It does this because mandating a single hierarchical partitioning of the world forces the user to know enough attributes to construct a path name according to that partitioning. We envision name servers sponsored by professional organizations (e.g., ACM special interest groups and IEEE societies), mailing lists, user groups, and so on. Furthermore, the loose coupling of name servers allows the confederation to evolve "bottom-up" without the need for centralized control. In such an environment, clients (users and system administrators) "subscribe" to those name servers that might be of use.

The X.500 naming system adopts an alternative design in which the attribute-based name space is restricted to a hierarchy. One reason for doing this is to support a unique "distinguished name" for each object. X.500 accommodates users that have incomplete information by associating a set of attributes with each object, and allowing users to "browse" the attributes at each node in the hierarchy. Thus, to learn the distinguished name of an object for which the user has incomplete information, the user traverses the hierarchy one node at a time, browsing through the attributes at each node.

5.3 Interface

The user interface provides a mechanism for using information learned from Profile to access resources by name. The most critical factor in the design of the user interface is the degree of confidence the user has in Profile returning attributes for the principal that the user "intended" to identify with the name. Our experience shows that there is a spectrum of styles for interacting with Profile. At one extreme, the user queries Profile to learn about a certain principal. This is generally the case when the principal in question is a remote user. At the other extreme, the user trusts that a certain profile name identifies a particular principal, and is willing to let Profile invoke a base command on the attributes returned by a name server. In practice, users tend to trust their aliases, and to some degree, the local name server and name servers that identify organizations. In the middle of the spectrum, a user might be willing to let the interface apply a base command to an argument, but only if given a chance to confirm or abort the action. A related issue is the sensitivity of the command; a user might be willing to list the files in the home directory of the wrong principal, but not willing to send mail to the wrong principal.

Although our experience with **pcsh** is limited to a small user community, there appears to be a legitimate use for a more sophisticated interface than just **profile_lookup** command, and **pcsh** offers a reasonable tool with which the user can build such an interface. As an example of how easily the tool allows the user to extend the interface, by registering an attribute with the tag **printer** for each user principal with a printer in their office and defining a specification for a **print** profile command, it is easy to send output to the printer in John Smith's office by typing

print "john" file.dvi

As another example, the tool has proved useful for building commands that take advantage of attributes registered for objects other than principals, e.g.,

compile "is_a=processor, architecture=68020, mips>2.0, load=min" file.c

Using a procedure-based translation mechanism to generate locally understood names proved to be a powerful and flexible technique for integrating the Profile name space with other naming systems. In particular, although the scope of the general problem is on the order of "number of commands × number of autonomous systems," our experience shows that a small number of attributes and templates are sufficient in practice. We also observe that the procedure-based approach seems more appropriate than the table-based approach exploited by Schwartz, Zahorjan, and Notkin [24]. The reason for this is that the naming systems in which we are interested are implicitly associated with commands; they do not exist as segregated naming services with well-defined interfaces.

6. CONCLUDING REMARKS

This paper describes the Profile naming service. The novel aspects of Profile are its attribute-based name servers, a flexible mechanism for searching a confederation name space, and tools that can be used to build a customized user interface. A prototype implementation demonstrates that Profile is a useful and effective mechanism for identifying principals in a limited testbed. By extrapolating from this experience, we conclude that Profile's confederation of attribute-based name servers is extendible to a full-scale internet.

Based on our positive experience with Profile, we have used its architecture as a blueprint for a general protocol used to query attribute-based name servers [18]. Using this protocol, we plan to expand the Profile testbed to additional sites, including extensive coverage of the NASA Science Internet. In doing so, we hope to be able to address the unanswered questions regarding the name space search strategies. We also hope to settle on an appropriate set of *conventions* that govern how the system is to be applied and used; e.g., define a standard set of attributes and establish reasonable protection policies. Finally, we plan to continue our effort to integrate dissimilar naming systems by generalizing the procedure-based translation mechanism exploited by Profile.

APPENDIX: GRAMMAR FOR ATTRIBUTES

This specification uses a subset of Backus–Naur Form that includes the following metacharacters.

|	Elements separated by bar ('|') are alternatives.
' '	Encloses literal text.
;	Starts a comment that continues to the end of the line.
::=	Assignment sign; the left side is a nonterminal and the right side is a rule.

```
attribute ::= expression
expression ::= attr_str
     | '[' expression ']'          ; optional
     | expression '|' expression   ; alternative
     | expression '*' expression   ; wildcard (named attribute only)
     | expression '?' expression   ; single character wildcard
                                     (named attribute only)
     | '(' expression ')'          ; expression itself
     | empty
attr_str ::= attr_substr | attr_substr blank attr_str
attr_substr ::= attr_alphabet | attr_alphabet attr_substr
attr_alphabet ::= letter | digit | attr_sc
attr_sc ::= ''' | '-' | '.' | '=' | '%' | '@' | ':' | '!' | '#' | '&' | ' '
letter ::= 'A' | . . . | 'Z' | 'a' | . . . | 'z'
digit ::= '0' | . . . | '9'
blank ::= ' '
                                   ; the blank character
empty ::= ''                       ; the empty string
```

ACKNOWLEDGMENTS

Herman Rao, Mic Bowman, and Gregg Townsend contributed to the implementation of Profile and provided valuable feedback. Norm Hutchinson, Richard Schlichting, and the referees provided valuable comments on earlier drafts of this paper; discussions with Dave Clark and Karen Sollins were also quite helpful. Norm Hutchinson and Doug Comer provided assistance in porting Profile to the University of Washington and Purdue University. Ralph Griswold made the mailing list for the ICON users group available for testing.

REFERENCES

1. AHO, A., KERNIGHAN, B., AND WEINBERGER, P. Awk—a pattern scanning and processing language. In *UNIX Programmers Manual*, 2, Berkeley Software Distribution, Berkeley, Calif., 1980.
2. ALMES, G., BLACK, A., LAZOWSKA, E., AND NOE, J. The Eden system: A technical review. *IEEE Trans. Softw. Eng. SE-11*, 1 (Jan. 1985), 43-58.
3. BIRRELL, A., JONES, M., AND WOBBER, E. A simple and efficient implementation for small databases. In *Proceedings of the Eleventh SIGOPS Symposium on Operating System Principles* (Austin, Tex., Nov. 8-11, 1987). ACM, New York, 1987, 149-154.
4. BOWMAN, M., PETERSON, L., AND RAO, H. Univers: A name server for the next generation internet. Tech Rep. TR 88-17, Univ. of Arizona, Tucson, Mar. 1988.

5. DEBRAY, S., AND PETERSON, L. Reasoning about naming systems. Tech Rep. TR 87-16, Univ. of Arizona, Tucson, July 1987.
6. GALLAIRE, H., MINKER, J., AND NICOLAS, J. Logic and databases: A deductive approach. *ACM Comput. Surv. 16*, 2 (June 1984), 153-186.
7. ISO. Information processing systems: Open systems interconnection—the directory—overview of concepts, models, and service. Draft International Standard ISO 9594-1:1988(E).
8. JOY, W. An introduction to the C shell. In *UNIX Programmers Manual, 2*, Berkeley Software Distribution, Berkeley, Calif., 1980.
9. KERNIGHAN, B., AND RITCHIE, D. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, N.J., 1978.
10. LAMPSON, B. Designing a global name service. In *Proceedings of the Fifth Annual ACM Symposium on the Principles of Distributed Computing* (Calgary, Aug. 11-13, 1986). ACM, New York, 1986, 1-10.
11. LANTZ, K., EDIGHOFFER, J., AND HITSON, B. Towards a universal directory service. In *Proceedings of the Fourth Annual ACM Symposium on the Principles of Distributed Computing* (Minaki, Aug. 5-7, 1985). ACM, New York, 1985, 250-260.
12. MILLS, D., AND BRAUN, H. The NSFNET backbone network. In *Proceedings of the ACM SIGCOMM '87 Workshop* (Aug. 1987). ACM, New York, 1987.
13. MOCKAPETRIS, P. Domain names—implementation and specification. Request for comments 1035. USC-ISI, Marina del Rey, Calif., Nov. 1987.
14. MYERS, E. An *O*(*ND*) difference algorithm and its variations. *Algorithmica 1*, 2 (1986), 251-266.
15. NEEDHAM, R., AND BIRRELL, A. The CAP filing system. In *Proceedings of the Sixth Symposium on Operating System Principles* (West Lafayette, Nov. 16-18, 1977), ACM, New York, 1977, 11-16.
16. OPPEN, D., AND DALAL, Y. The Clearinghouse: A decentralized agent for locating named objects in a distributed environment. Tech. Rep. OPD-T8103, Xerox Office Products Div., Palo Alto, Calif., Oct. 1981.
17. PETERSON, L. A yellow-pages service for a local-area network. In *Proceedings of the ACM SIGCOMM '87 Workshop* (Aug. 1987). ACM, New York, 1987, 235-242.
18. PETERSON, L., AND SOLLINS, K. The universal naming protocol. Draft request for comments, June 1988.
19. PICKENS, J., FEINLER, E., AND MATHIS, J. The NIC name server—a datagram based information utility. In *Proceedings of the Fourth Berkeley Workshop on Distributed Data Management and Computer Networks* (Aug. 1979).
20. POSTEL, J. File transfer protocol. Request for comments 765. USC-ISI, Marina del Rey, Calif., June 1980.
21. POSTEL, J., SUNSHINE, C., AND COHEN, D. The ARPA internet protocol. *Comput. Netw. 5* (May 1981), 261-271.
22. REITER, R. On closed world databases. In *Logic and Databases*. H. Gallaire and J. Minker, Eds., Plenum Press, New York, 1987, 55-76.
23. SALTZER, J., AND SCHROEDER, M. The protection of information in computer systems. *Proc. IEEE 63*, 9 (Step. 1975), 1278-1308.
24. SCHWARTZ, M., ZAHORJAN, J., AND NOTKIN, D. A name service for evolving heterogeneous systems. In *Proceedings of Eleventh Symposium on Operating System Principles* (Nov. 1987), 52-62.
25. SOLLINS, K. Distributed name management. Ph.D. dissertation, MIT, Feb. 1985.
26. SOLOMON, M., LANDWEBER, L., AND NEUHENGEN, D. The CSNET name server. *Comput. Netw. 6* (1982), 161-172.

Received October 1987; revised June 1988; accepted June 1988